**Critical Thinking Assignment 1:** **Hello Android App**

Alexander Ricciardi

Colorado State University Global

CSC475: Platform-Based Development

Dr. Herbert Pensado

February 16, 2025

**Critical Thinking Assignment 1: Hello Android App**

This documentation is part of the Critical Thinking 1 Assignment from CSC475: Platform-Based Development at Colorado State University Global. The documentation provides an overview of the Android application's functionality and testing scenarios including application pseudocode, Kotlin code lines, and the application’s output screenshots. It also reflects on the obstacles faced during the application’s development, and the skills acquired. The application is coded in Kotlin 1.9.24 and XML, and is named “Simple Hello Android App.”

**The Assignment Direction:**

Option #1: "Hello Android"

Create a simple Android application that displays a "Hello, Android!" message on the screen when launched. Familiarize yourself with the basic structure of an Android app and practice using Kotlin syntax.

Please ensure that your submission includes the following components:

Source code file(s) containing the program implementation.

A 1-page paper explaining the program's purpose, the obstacles faced during its development, and the skills acquired. The paper should also include screenshots showcasing the successful execution of the program.

Compile and submit your pseudocode, source code, and screenshots of the application executing the application, the results and GIT repository in a single document.

**Program Description:**

This is a simple Hello Android Application written in Kotlin. It displays a simple animation where a TextView ("Hello Android!") bounces around within the screen's boundaries. It also provides a toggle button allowing the user to stop and restart the text animation.

**⚠️Note:**

* The program uses a background thread to run a text animation within an infinite loop.  
  This loop updates the TextView's position and, when hitting a screen boundary the text bounces and changes color. This ‘hands-on’ animation approach is implemented for learning purposes, it is generally better and good practice to use Android API’s built‑in animation classes (like those in AnimationUtils).
* To initialize the application, I used the Empty View Activity Template from Android Studio. Then I modify the file to implement the text animation and my own icon.

**Git Repository:**

This is a picture of my GitHub page:

A screenshot of a computer

AI-generated content may be incorrect.

I use [GitHub](https://github.com/) as my Distributed Version Control System (DVCS), the following is a link to my GitHub, [Omegapy](https://github.com/Omegapy).

My GitHub repository that is used to store this assignment is named [My-Academics-Portfolio](https://github.com/Omegapy/My-Academics-Portfolio) and the link to this specific assignment is: https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Platform-Based-Dev-Android-CSC475/Module-1-Critical-Thinking

**Project Map:**

* CTA1 Hello Android App.docx (this file, App documentation)
* MainActivityPseudo.txt (Main Activity pseudocode)

The project used files from the Android Studio’s *Empty View Activity* template. Additionally, only the template files that were modified to accommodate the functionality of the application are listed below:

* MainActivity.kt (Kotlin code, application logic)
* Main\_activity.xml (XML code, main UI layout)
* string.xml (resource file storing strings)

The following files have been overridden to accommodate my icon. If you do not want to use my logo, do not use these files. The template will automatically use the Android icon.

* color.xml
* drawable
  + ic\_launcher\_background.xml
  + ic\_launcher\_foreground.xml
* mipmap
  + ic\_launcher.xml
  + ic\_launcher\_foreground.xml
  + ic\_launcher\_round.xml

**Reflection**

This is my first time using Kotlin and Android Studio to create a program. Installing Android Studio was straightforward. Learning about Kotlin from the textbook and doing research about it was rewarding. The following is an overview of Kotlin based on what I have learned from the textbook and my research.

Kotlin is often described as the most succinct language, meaning that is the least error-prone programming language (Horton, 2019). Android SDK (Software Development Kit) is written in Java. Moreover, Kotlin is fully interoperable with Java, meaning that Java libraries and frameworks can be easily integrated into Kotlin code, and Java projects also can be easily migrated to Kotlin.

* It is an Object-Oriented Programming language (OOP).
* It includes null safety, preventing null pointers.
* It allows function extensions, that is adding functionality to existing classes without modifying their source code.
* It allows data classes which are classes primarily used to hold data.
* It can implement coroutines, making asynchronous programming much easier such as handling network requests.

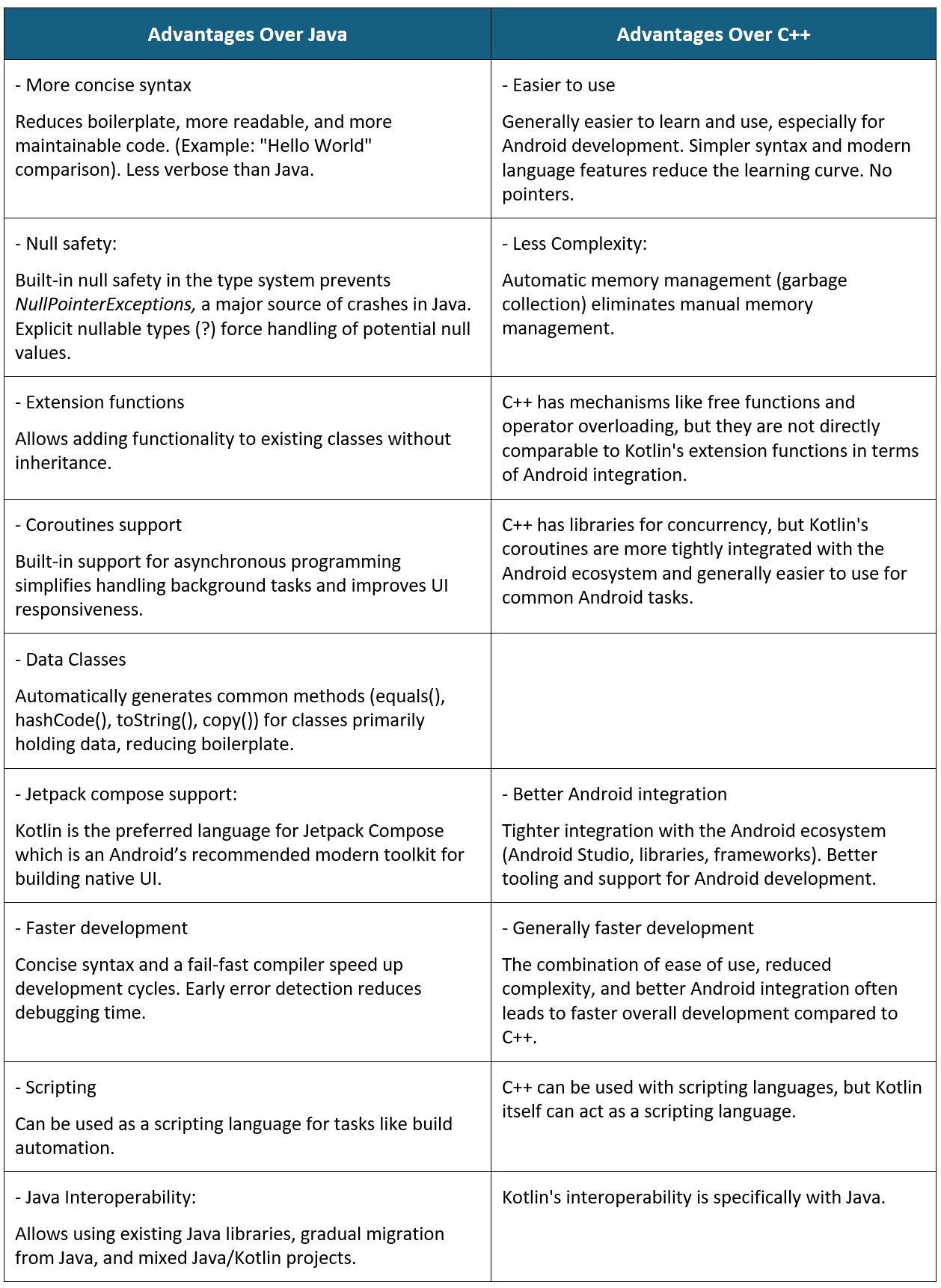
(Ricciardi, 2025)

Furthermore, it is generally preferred for Android app development over programming languages like C++ and Java. The table below lists some of the advantages Kotlin has over C++ and Java.

*See the next page.*

**Table 1**

*Kotlin’s advantages over C++ and Java*



*Note:* The table lists the advantages that Kotlin has over C++ and Java when developing Android based Apps. From “Why Kotlin is preferred for Android app development” by Ricciardi (2025).

As described previously, I chose to create a “Simple Hello Android APP” by modifying the *Empty View Activity* Template from Android Studio and implementing a background thread to run a text animation within an infinite loop. This loop updates the *TextView*'s position and, when the text hits a screen boundary the text bounces and changes color. This was done to practice implementing variables and loops using Kotlin.

**Kotlin Variables Handling**

Kotlin uses type inference, type check, and smart-cast, meaning that, when type referring, the compiler automatically deduces the data types of variables and expressions, eliminating the need for explicitly declaring variable types (App Dev Insights, 2023). Type checks are performed in two ways by using the ‘*is*’ operator and its negation ‘*!is*’. These operations return a Boolean, that is return true if the condition is met. For example, if a variable ‘is’ of the data typed checked the condition will return true if not it will return false. Smart-cast allows the compiler to cast a variable to a specific type within a code block, without the need for explicit type checks and casts; for example, print(x.length) // x (a string) is automatically cast to String.

Furthermore, Kotlin uses two different keywords to declare variables, ‘*val*’ and ‘*var*’ (Developers, n.d.). ‘*val*’ is used to declare static variables, variables that never change, and ‘*var*’ is used to declare non-static variables, variables whose value can change. Kotlin's syntax is very similar to Java, and transitioning from Java to Kotlin for this assignment was traits forward. The part that was most challenging for me was learning the Android Project Structure, that is how the project files are interconnected using the Android Manifest file and Gradle build configurations

**My Android** **Project File Structure**

Learning my project file structure dependencies was essential for me to understand how different parts of the application interact and to implement a functional code. Below is a breakdown of the key components and their description:

Android

└── app <-- The main application module.

│ ├── manifests

│ │ └── AndroidManifest.xml <-- Configuration file for the app (permissions, activities, etc., not modified).

│ ├── kotlin+java <-- Contains the source code (Kotlin).

│ │ ├── com.example.helloandriod <-- Main package

│ │ │ └── MainActivity.kt <-- Activity class (Kotlin file, modified for this project).

│ │ ├── com.example.helloandriod (androidTest) <-- Instrumented tests (run on a device/emulator).

│ │ └── com.example.helloandriod (test) <-- Local unit tests (run on the JVM).

│ └── res <-- Resources for application (layouts, images, strings, etc.).

│ │ ├── drawable <-- Image files (PNG, JPG, XML drawables).

│ │ │ ├── ic\_launcher\_background.xml <-- Background for adaptive launcher icon (override for this project)

│ │ │ └── ic\_launcher\_foreground.xml <-- Foreground for adaptive launcher icon (override for this project)

│ │ ├── layout <-- XML files that define the UI layouts for your activities/fragments.

│ │ │ └── activity\_main.xml <-- Layout for your MainActivity (modified)

│ │ ├── mipmap <-- App icon files (different resolutions for different screen densities).

│ │ │ ├── ic\_launcher (6) <-- Regular app icon (multiple resolutions, override for this project).

│ │ │ ├── ic\_launcher\_foreground (5) <-- Adaptive icon foreground (multiple resolutions, override for this project).

│ │ │ ├── ic\_launcher\_round (6) <-- Round app icon (multiple resolutions, override for this project).

│ │ └── values <-- Resource files for values like colors, strings, and styles.

│ │ │ ├── colors.xml <-- Color definitions (modified for this project).

│ │ │ ├── strings.xml <-- String definitions (modified for this project).

│ │ │ └── themes (2) <-- Theme definitions (light and dark/nigh, not modified for this project.).

│ │ │ │ ├── themes.xml

│ │ │ │ └── themes.xml (night)

│ │ │ └── xml

│ │ │ │ ├── backup\_rules.xml

│ │ │ │ └── data\_extraction\_rules.xml

│ │ │ └── res (generated)

└── Gradle Scripts <-- Gradle build system (not modified).

Note that the *MainActivity.kt* and *activity****\_****main.xml* are the core files of the application.

* The *MainActivity.kt* file contains the Kotlin source code dictating the logic for the application, in this example the logic for the bouncing text background animation thread and the toggle button.
* The *activity****\_****main.xml* contains the XML code that defines the user interface layout, including the *TextView* for the "Hello Android!" text and the *ToggleButton*.
* The *ic\_launcher* files have been modified or overridden to accommodate my personal icon.
* The *color.xml* file has been modified to integrate a color background for my icon.
* The *strings.xml* file has been modified to store the *ToggleButton* text describing its state and the *bouncing\_text* (“Hello Android!”)

**Pseudocode and Koltin Code**

Implementing the animation of the bouncing text was not very difficult as I had implemented this functionality in other programming languages, more specifically Java and Python. The most difficult part was understanding how to get the elements from the UI so they could be integrated into the application logic (*MainActivity.kt*). The Android environment uses ‘*id*’ to accomplish it. For example, to get the size of the screen (‘*View*’) to set the boundaries of the text animation, I use the ‘*id’* of the element ‘*View*’ from the *activity****\_****main.xml* layout file after I imported it and set it as the application content view using this following line of code setContentView(R.layout.activity\_main) Below is a Kotlin code snippet demonstrating how this process is done:  
  
**Code Snippet 1**

*Using Ids to Connect Logic and UI*

// Set content view, the UI from the activity\_main.xml layout file  
setContentView(R.layout.activity\_main)  
  
// Layout's views attribute from activity\_main.xml layout file  
val container = findViewById<View>(R.id.main) // The root container (ConstraintLayout)  
val textView = findViewById<TextView>(R.id.bouncingText) // The TextView that will bounce  
// Button to start/stop the animation  
val toggleButton = findViewById<Button>(R.id.toggleButton)

*Note:* The Kotlin code snippet illustrated how UI (layout) and Logic are connected using ids. This code is part of the *onCreate* function.

The other challenging part was understanding the functionality of the

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) method within the *onCreate* function, to keep it simple, the code functionality avoids overlaps between window insets (UI element of the app behind system elements) with system UI elements (e.g. Status Bar, Navigation Bar). If the app's content overlaps with the system UI elements, it can lead to poor user experience.

**Figure 1**  
*Main Activity Pseudocode*

A black and white screen with text

AI-generated content may be incorrect.

*Note:* The figure illustrates a snapshot of pseudocode implementing the logic of a text view bouncing and changing color within a container (presumably a screen). The animation runs in a background thread. Additionally, the code implements a toggle button to stop and restart the animation. Furthermore, this pseudocode was created with an Android application perspective in mind and is intended to be translated into Kotlin. Please see *MainActivityPseudo.txt* file for the complete code.

**Figure 2**  
*Main Activity Kotlin (MainActivity.kt)*

A screen shot of a computer

AI-generated content may be incorrect.

*Note:* The figure illustrates a snapshot of Kotlin code implementing the logic of a text view bouncing and changing color within a container (a screen). The animation runs in a background thread. Additionally, the code implements the logic of a toggle button to stop and restart the animation. Please see *MainActivity.kt* file for the entire code.

**Code Snippet 4**  
*Main Activity XLM Layout UI (main\_activity.xml)*

<?xml version="1.0" encoding="utf-8"?>  
<!--  
===================================================================================================  
This file is a resource file written in XML that defines the user interface layout,  
including the TextView for the "Hello Android!" text and the ToggleButton.  
  
It is part of the Simple Hello Android App  
Author: Alexander Ricciardi  
Date: 02/14/2025  
  
Requirement: Kotlin and XML  
  
Program Description:  
 This is a simple Hello Android Application written in Kotlin.  
 It displays a simple animation where a TextView ("Hello Android!") bounces around within  
 the screen's boundaries.  
 It also provides a toggle button allowing the user to stop and restart the text animation.  
==================================================================================================  
-->  
<androidx.constraintlayout.widget.ConstraintLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <TextView  
 android:id="@+id/bouncingText"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@string/bouncing\_text"  
 android:textSize="24sp"  
 app:layout\_constraintBottom\_toTopOf="@+id/toggleButton"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <Button  
 android:id="@+id/toggleButton"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@string/toggle\_button\_state1"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 android:layout\_marginBottom="16dp" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

*Note:* The XML snippet illustrates the UI layout of the application. Please also see *main\_activity.xml* file for the code.

To connect the elements from the UI, this file (*main\_activity.xml*), to the application logic, Android uses elements’ ‘*id*s’; for example the ‘*TextView*’ element as an ‘*id*’ associated with ‘*bouncingText*’ as illustrated by the following code lines:

<TextView  
 android:id="@+id/bouncingText"

The following example illustrates how the string “*Hello Android!*” is handled within the Android ecosystem to implement the application functionality.

*string.xml* file:  
<string name="bouncing\_text">Hello Android!</string>

*main\_activity.xml* file:

<TextView

android:id="@+id/bouncingText"

android:text="@string/bouncing\_text"

The *‘@string*’ refers to the *string.xml* file

*MainActivity.kt* file:

setContentView(R.layout.activity\_main)

val textView = findViewById<TextView>(R.id.bouncingText)

Note that the ‘*textView*’ is a ‘*TextView*’ object and the string ‘*Hello Android!*’ is encapsulated within the object. To get the encapsulated string we can use two methods:

* Method-1: val text = textView.text.toString() this method casts the ‘*textView*’ object to a string.
* Methode-2: val text = textView.getString(R.string.boucing\_text) this method gets the string that is associated with the *string.xml* file, if the string is changed within the ‘*textView*’ object, this method will not reflect it. It is safer and good practice to use the first method-1 for ‘*TextView*’ object. On the other hand, for objects like the *‘Button*’ type (e.g.toggle-button.text = getString(R.string.toggle\_button\_state2)) as the encapsulated text will not change, it is static, using method-2 is adequate.

*See next page*

**Screenshots**

This section demonstrates the output of the applications using GIF format images.

**Figure 3**

*Bouncing Text and Toggle Button*

A screenshot of a cell phone

AI-generated content may be incorrect.

*Note**:* The figure GIF animation depicts the ‘*Hello Android*!’ text bouncing and changing color, from black to dark green after bouncing off the edge of the screen. It also depicts the user clicking on the toggle button to stop and restart the animation. Note that the button text all changes from ‘*Stop Animation*’ to ‘*Start Animation*’ when first clicked and then from ‘*Start Animation*’ to ‘Stop Animation’ when clicked again.

*See next page*

**Figure 4**

*Icon and Home Buttons*

A screenshot of a cell phone

AI-generated content may be incorrect.

*Note:* The figure GIF animation depicts a user using the ‘Home’ and the application icon buttons. Note that the animation seems not to be fluid, this is due to the GIF animation looping, not the application animation itself.

**Summary**

The "Simple Hello Android App" project provided me with a good foundation for understanding Android development using Kotlin and XML. The process of creating the bouncing text animation and the toggle button helped me to become more familiar with Kotlin, how it is used to implement the application logic (MainActivity.kt), and how it interconnects with the XML UI design (using *ConstraintLayout*, *TextView*, and *Button* in *activity\_main.xml*), and the application resources (e.g. *strings.xml*). Understanding the Android project structure, connecting UI elements to Kotlin code via resource ‘ids’, and managing a background thread for animation, were the most challenging parts of the project. Ultimately, the project provided me with a solid grasp of fundamental Android concepts and its ecosystem.

**References:**

App Dev Insights. (2023, December 16). Kotlin Type inference, Type check, Smartcast - App Dev Insights - Medium. *Medium*. https://medium.com/@appdevinsights/kotlin-type-inference-type-check-smartcast-49c5f98fac1b

Developers (n.d.). *Learn the Kotlin programming language*. Android. https://developer.android.com/kotlin/learn

Horton, J. (2019). Android programming with Kotlin for beginners. Packt Publishing. ISBN: 9781789615401

Ricciardi, A. (2025, February 14). *Why Kotlin is preferred for Android app development*. Omegapy │ Code Chronicles. https://www.alexomegapy.com/post/why-kotlin-is-preferred-for-android-app-development